Рекомендації з написання PHP коду

Вітаю! Сьогодні розглянемо основні рекомендації щодо написання чистого та ефективного коду PHP. Ці рекомендації допоможуть вам писати код, який легко читати, зрозуміти та підтримувати.

Шеремет Андрій Григорович
ПЗПІ-22-6

13.12.2024

Мета презентації

Ознайомити студентів з основними рекомендаціями щодо написання чистого, ефективного та підтримуваного коду для PHP, а також навчити аналізувати та рефакторити код для покращення його якості.

Що таке Code Conventions?

Code Conventions - це набір правил та рекомендацій щодо написання коду. Вони охоплюють стиль написання, іменування змінних, функцій та класів, а також форматування коду.

1 **Стиль коду**

Відступи, пробіли, розбиття на рядки.

2 **Іменування**

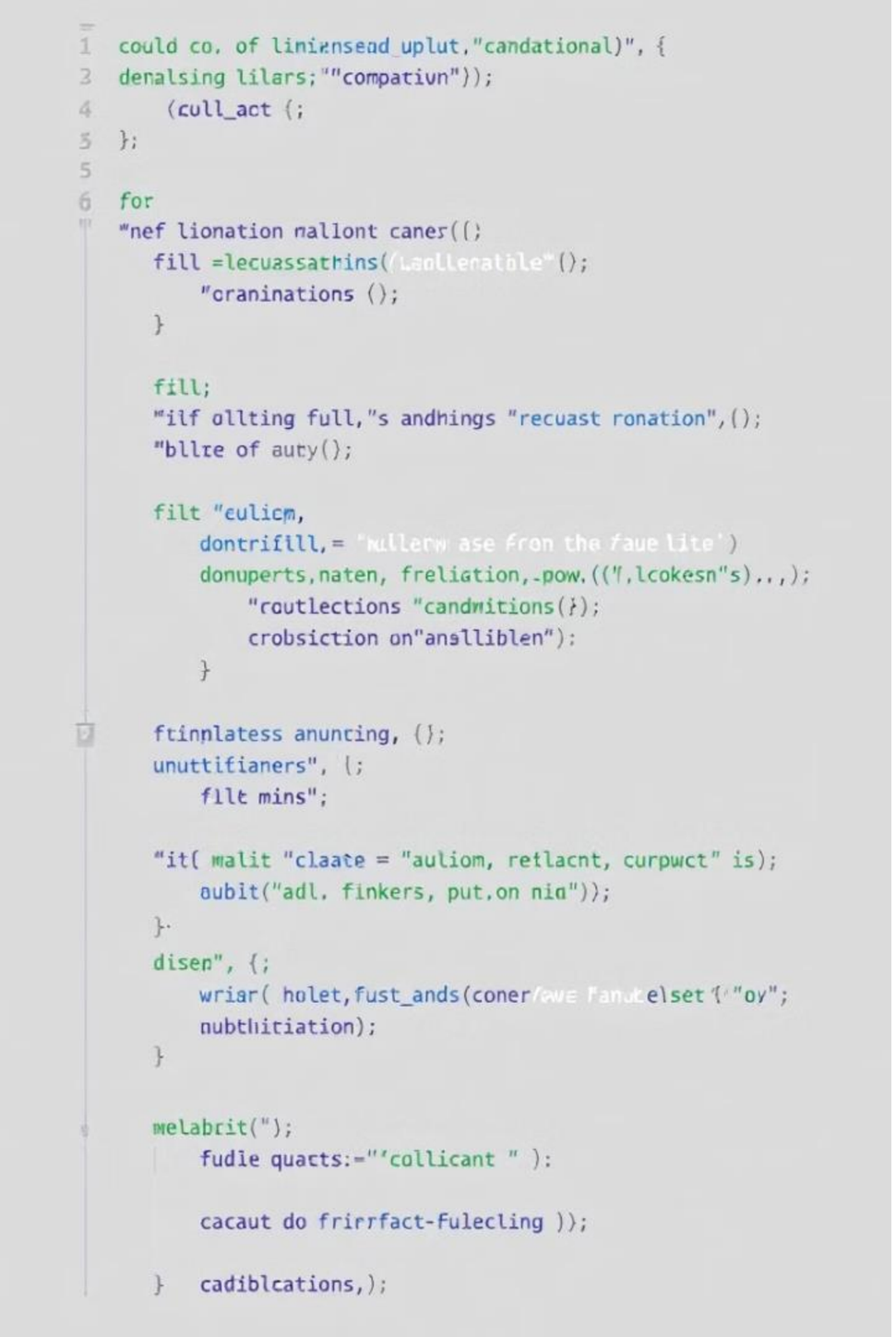
Вибір назв для змінних, функцій функцій та класів.

3 **Коментарі**

Які коментарі додавати до коду та як їх формулювати.

4 **Організація**

Структурування коду на файли та папки.



Вступ

Обрана мова програмування: PHP

PHP — це популярна серверна мова програмування, яка використовується для створення вебдодатків. Вона була створена спеціально для роботи з HTML і чудово підходить для динамічних вебсайтів.

Динамічна типізація

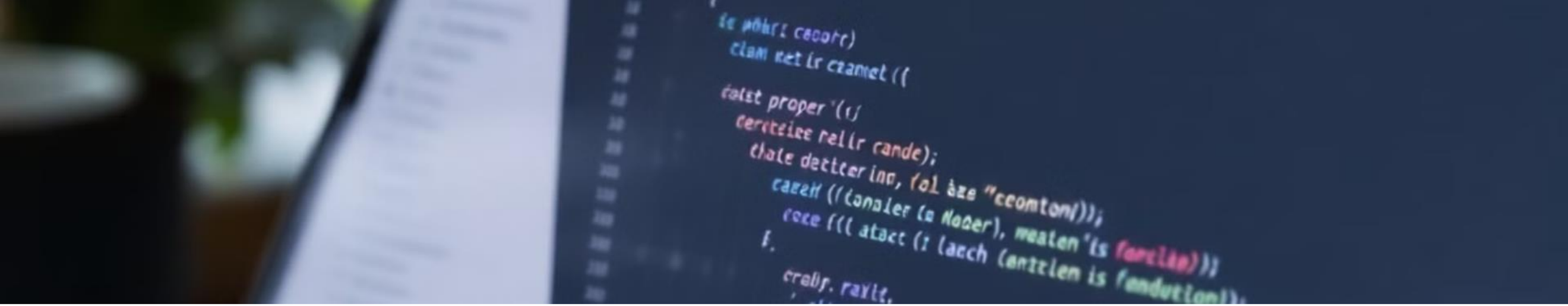
Дозволяє писати код швидко, але вимагає уваги до типів даних.

Вбудована підтримка баз даних

Має зручні інструменти для роботи з MySQL, PostgreSQL, SQLite тощо.

Широка екосистема

фреймворки (Laravel, Symfony), CMS (WordPress, Drupal) та бібліотеки.



Основні рекомендації

Розглянемо основні рекомендації, які допоможуть вам написати якісний та ефективний код PHP.

Стиль коду

1 Відступи

Використовуйте 4 пробіли для відступів, не табуляцію

2 Дужки та пробіли

Після ключових слів **if, for, while, switch, function** обов’язково ставте пробіл перед дужками

Відкриваюча дужка { для функції, класів, методів повинна бути на тій самій лінії, що й декларація

3 Довжина рядка

Максимальна довжина рядка — 120 символів. Для кращої читабельності рекомендовано обмежувати рядки до 80 символів.

Приклад коду:

```php
<?php

function example() {
    if ($condition) {
        echo 'Hello!';
    }
}
```

Іменування

1 **camelCase для функцій**

Використовуйте camelCase для імен змінних і функцій

2 **PascalCase для класів**

Використовуйте PascalCase для назв класів

3 **Константи**

Константи пишіть великими літерами з підкресленнями

Приклад коду:

```php
<?php

define('API_KEY', '12345');

class UserController {
    const MAX_LIMIT = 100;
    function getUserData() {
        $userName = 'John';
    }
}
```

Структура коду

1 **Модулі**

Розбивайте код на модулі та класи для полегшення підтримки та повторного використання.

2 **1 - 1**

Дотримуйтеся принципу одна функція/метод — одна одна задача.

3 **Групування**

Групуйте логічно пов’язані функції та методи разом у класах або неймспейсах.

Приклад коду:

```php
<?php
namespace App\Controllers;

class UserController {
    public function create() {
        // ...
    }
    public function update() {
        // ...
    }
}
```

Принципи рефакторингу

1 Метод повинен виконувати лише одну дію

Якщо метод виконує декілька завдань, його слід розбити на кілька менших методів. Це покращує читабельність та спрощує повторне використання коду.

2 Уникайте дублювання коду

Виносьте повторювані фрагменти у функції або окремі класи. Зменшення дублювання полегшує підтримку коду — змінювати потрібно лише одне місце.

3 Заміна магічних чисел і рядків на константи

Використовуйте зрозумілі константи замість «магічних» чисел і рядків у коді. Це робить код зрозумілішим, а також дозволяє легко змінювати значення констант у майбутньому.

Приклад коду:

```php
function getUserById($userId) {
    return Database::fetchUser($userId);
}

function calculateDiscount($price, $discountPercentage) {
    return $price * ($discountPercentage / 100);
}

const DEFAULT_DISCOUNT = 10;

function applyDiscount($price) {
    return calculateDiscount($price, DEFAULT_DISCOUNT);
}
```

Оптимізація продуктивності

1 **Використовуйте кешування**

Кешуйте результати часто виконуваних операцій, таких як запити до бази даних або обробка даних. Зменшується навантаження на сервер і прискорюється виконання коду.

2 **Уникайте зайвих обчислень у циклах**

Виносьте незмінні обчислення за межі циклу, щоб уникнути їх повторного виконання. Це зменшує кількість операцій, які виконуються під час кожної ітерації циклу, що прискорює роботу скрипта

3 **Працюйте з великими обсягами даних частинами**

Для великих масивів даних використовуйте ітератори або обробку даних частинами, щоб уникнути перевантаження пам’яті. Це дозволяє економити оперативну пам’ять і уникати перевищення лімітів PHP.

Приклад коду:

```php
function getCachedUserData($userId) {
    static $cache = [];

    if (!isset($cache[$userId])) {
        $cache[$userId] = Database::fetchUser($userId);
    }

    return $cache[$userId];
}

$totalUsers = count($users);
for ($i = 0; $i < $totalUsers; $i++) {
    echo $users[$i]['name'];
}

$handle = fopen('large_file.csv', 'r');
while (($line = fgetcsv($handle)) !== false) {
    processLine($line);
}
fclose($handle);
```

Обробка помилок

1 **Використовуйте виключення (try-catch) Замість die(), exit()**

Замість примусової зупинки скрипта використовуйте використовуйте механізм обробки виключень. Це дозволяє програмі продовжувати роботу або коректно завершити виконання після обробки помилки.

2 **Створюйте власні класи виключень**

Для специфічних типів помилок створюйте власні класи, що успадковують Exception. Це дозволяє гнучкіше обробляти різні типи помилок у коді.

3 **Логування помилок**

Використовуйте логування для запису помилок у файли чи зовнішні системи моніторингу. Логи дозволяють аналізувати причини помилок та швидко їх виправляти.

Приклад коду:

```php
function divide($a, $b) {
    if ($b === 0) {
        throw new Exception("Ділення на нуль неможливе");
    }
    return $a / $b;
}

try {
    echo divide(10, 0);
} catch (Exception $e) {
    echo "Помилка: " . $e->getMessage();
}

class DatabaseException extends Exception {}

function connectToDatabase() {
    throw new DatabaseException("Не вдалося підключитися до бази даних");
}

try {
    connectToDatabase();
} catch (DatabaseException $e) {
    echo "Помилка бази даних: " . $e->getMessage();
}

function logError($message) {
    error_log($message, 3, 'errors.log');
}

try {
    connectToDatabase();
} catch (DatabaseException $e) {
    logError("[" . date('Y-m-d H:i:s') . "] " . $e->getMessage() . "\n");
}
```

Дотримання парадигм програмування

1 Дотримуйтесь принципів ООП

Створюйте класи з чітко визначеними відповідальностями, використовуйте інкапсуляцію, успадкування та поліморфізм. Це дозволяє писати зрозумілий, гнучкий та масштабований код.

2 Використовуйте функціональний підхід

Використовуйте анонімні функції, функції першого класу та функції вищого порядку для створення гнучкого коду. Функціональний стиль допомагає зменшити кількість побічних ефектів і підвищити тестованість коду.

3 Дотримуйтесь принципу SOLID

Пишіть код, який відповідає принципам SOLID (єдине завдання, відкритість-закритість, підстановки Лісков, розділення інтерфейсів, інверсія залежностей). Це забезпечує легкість у підтримці та розширенні коду.

Приклад коду:

```php
class User {
    private $name;

    public function __construct($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }
}

$user = new User("John");
echo $user->getName();

$numbers = [1, 2, 3, 4, 5];
$squaredNumbers = array_map(fn($n) => $n * $n, $numbers);
print_r($squaredNumbers);

class EmailSender {
    public function sendEmail($email, $message) {
        echo "Email sent to $email with message: $message";
    }
}

class UserRegistration {
    private $emailSender;

    public function __construct(EmailSender $emailSender) {
        $this->emailSender = $emailSender;
    }

    public function register($email) {
        $this->emailSender->sendEmail($email, "Welcome!");
    }
}
```

Тестування та документування коду

1 Пишіть модульні тести

Використовуйте бібліотеки для тестування, такі як PHPUnit, щоб створювати модульні тести для окремих функцій та класів. Тести дозволяють впевнено змінювати код, перевіряючи його коректність після кожної зміни.

2 Дотримуйтесь принципу TDD

Спочатку пишіть тести, а потім — код, що їх проходить. Цей підхід допомагає уникнути зайвого коду та гарантує, що всі функції працюють відповідно до очікувань.

3 Документуйте код коментарями та використовуйте PHPDoc

Використовуйте стандарт PHPDoc для документування функцій, методів, параметрів і типів даних. Документація полегшує розуміння коду іншими розробниками та спрощує підтримку проєкту.

Приклад коду:

```php
use PHPUnit\Framework\TestCase;

class Calculator {
    public function add($a, $b) {
        return $a + $b;
    }
}

class CalculatorTest extends TestCase {
    public function testAdd() {
        $calculator = new Calculator();
        $this->assertEquals(5, $calculator->add(2, 3));
    }
}

/**
 * Додає два числа.
 *
 * @param int $a Перше число
 * @param int $b Друге число
 * @return int Сума чисел
 */
function addNumbers(int $a, int $b): int {
    return $a + $b;
}

echo addNumbers(3, 7);
```

Приклад неправильно написаного коду

Неохайний код без відступів, неправильне іменування.

```php
<?php
const $usertoken;
class user_service {
function GetuserData($id){
$userdata=['id'=>$id,'name'=>'John Doe','email'=>'john.doe@example.com'];return $userdata;
}

function UPDATEuserdata($id,$data){
    // Логіка оновлення
return true;}
}
```

Приклад правильно написаного коду

Чистий код з відступами, чіткими назвами змінних.

```php
<?php
declare(strict_types=1);

namespace App\Services;

/**
 * Клас UserService для роботи з користувачами.
 */
class UserService
{
    /**
     * Отримати дані користувача за його ідентифікатором.
     *
     * @param int $userId Ідентифікатор користувача.
     * @return array Дані користувача.
     */
    public function getUserData(int $userId): array
    {
        // Імітація даних користувача
        $userData = [
            'id' => $userId,
            'name' => 'John Doe',
            'email' => 'john.doe@example.com',
        ];

        return $userData;
    }

    /**
     * Оновити дані користувача.
     *
     * @param int $userId Ідентифікатор користувача.
     * @param array $data Дані для оновлення.
     * @return bool Результат оновлення.
     */
    public function updateUserData(int $userId, array $data): bool
    {
        // Тут могла б бути логіка оновлення даних у базі
        // Повертаємо успішний результат
        return true;
    }
}
```

Роль Code Conventions у командній роботі

В команді важливо дотримуватися однакових правил написання коду. Це спрощує комунікацію, зменшує кількість помилок та покращує ефективність розробки.

1

Спільне розуміння

Всі члени команди розуміють структуру коду.

2

Легше обслуговувати

Простіше підтримувати та змінювати код у майбутньому.

3

Зменшення помилок

Спільне дотримання правил зменшує кількість помилок.

Висновки

Запропоновані рекомендації щодо написання коду на PHP, такі як дотримання правил іменування, принципів рефакторингу, оптимізації продуктивності, обробки помилок, дотримання парадигм програмування та правильного тестування і документування, значно покращують якість коду та спрощують роботу з ним.

1

Чіткий код

Код, який легко читати та зрозуміти.

2

Ефективний код

Код, який працює швидко та без помилок.

3

Зручний код

Код, який легко підтримувати та змінювати.